Learning complex games through self play - Pokémon battles

by

Miquel Llobet Sanchez

Submitted to the Barcelona School of Informatics
in partial fulfillment of the requirements for the degree of

Bachelor in Informatics Engineering

at the

POLYTECHNIC UNIVERSITY OF CATALONIA

June 2018

Author .....
Barcelona School of Informatics
June 29, 2018

Certified by .....
Alberto Cabellos Aparicio
Associate Professor - FIB
Thesis Supervisor

Certified by .....
Marta Arias Vicente
Associate Professor - FIB
Thesis Supervisor

Certified by .....
Xavier Giró i Nieto
Associate Professor - ETSETB
Thesis Supervisor

Abstract

English

In this project we analyze the feasibility of using reinforcement learning and self-play to train an agent playing Pokémon Battles. The game is analyzed in depth and it's unique properties and challenges revealed. The project surveys different reinforcement learning libraries and implements a simple multi-agent environment for a bleeding-edge research platform.

Català

En aquest projecte s'analitza la viabilitat d'utilitzar aprenentatge per reforç i "self-play" per entrenar un agent a jugar Batalles Pokémon. El joc és analitzat en detall i les seves propietats úniques són revelades. El projecte analitza diverses plataformes d'aprenentatge per reforç i implementa un entorn multi-agent senzill a una plataforma de recerca puntera.

Castellano

En este proyecto se analiza la viabilidad de utilizar aprendizaje por refuerzo y "self-play" para entrenar un agente a jugar Batallas Pokémon. El juego es analizado en detalla y sus propiedades únicas son reveladas. El proyecto analiza diversas plataformas de aprendizaje por refuerzo y implementa un entorno multi-agente sencillo en una plataforma de investigación puntera.

Contents

Chapter 1

Context and scope of the project

1.1 Context and problem formulation

In recent years, Reinforcement Learning agents have achieved superhuman performance in some of the most challenging classical games. This was initially accomplished with Backgammon [1] in 2002, and more recently a subset of ATARI games [2] and the game of Go by AlphaGo [3]. Due to compounded improvements in Deep Learning and the availability of cheap computational power, previously intractable problems are now within reach.

This new generation of agents combines the power of supervised learning with Deep Neural Networks and Reinforcement Learning (RL) - a method used to train agents to maximize rewards by interacting with an environment. Supervised learning is used with game replay data (state, actions, and reward tuples) to approximate functions used in various RL algorithms.

In complex games such as Go, researchers have relied extensively on expert game datasets for training - this high quality data is often hard to obtain, and not generalizable across problems. A new approach has emerged to overcome this challenge, self-play, where agents are trained by playing against versions of each-other. This has resulted in outstanding results: AlphaGo was recently beaten by AlphagoZero [4], an agent trained through self-play with zero knowledge of the game. More recently a self-trained Chess agent [5] surpassed Stockfish, state of the art in computational

Figure 1-1: Pokémon game mechanics - this work will focus on developing an agent to play in battles.

chess.

This approach has been successful in well-defined perfect information games, but top research labs are now moving to more complex games such as Starcraft II (Google DeepMind) [6] and Dota (OpenAI) [7], where intelligent agents are already beating the world's best players in constrained game modes. This project aims to expand on the subject by learning how to play a complex turn-based through self play: Pokémon battles 1.1.

The game of Pokémon [8] consists of capturing and training a variety of fictional creatures called "Pokémon" and using them to battle other trainers. In battles, trainers use up to 6 creatures, each with 4 possible moves with the objective of defeating the opposing team: reducing all the creatures' health points to zero. The game has a strong competitive scene with official world tournaments, but most of the competitive online play is done in PokemonShowdown [9], an online platform with tens of thousands of daily active players.

The game has some unique properties that make it a challenging problem for computers to solve. States are partially observable: players can only see parts of the opponent team and moves, increasing the branching factor considerably. Turns are atomic: actions can be executed in a different game state than observed. The game is also stochastic: action effectiveness is random in nature and risk has to be taken into account by players. There is no simple state evaluation metric: Pokemon health values are generally representative, but many moves in the game trade off damage for longer term status changes vital to strategy. Finally, simulations are costly: each turn

takes 5-40ms depending on complexity, which prohibits extensive state exploration within time constraints.

Experienced players show complex strategies that result in a combination of moves and Pokémon types and abilities that so far other research hasn't been able to replicate. Strategies vary from using hyper offensive teams ("sweepers"), focused on knocking out the opposing team quickly, or "Wall" teams, focused on resisting heavy attackers and slowly damaging the opponent. Overall there are different moves that are preferred in the competitive setting, such as Spikes, which can decrease Pokémon's health significantly at the start of a turn. Success in this project will result in the agent learning policies that exploit these moves and strategy combinations.

1.2 State of the art

There have been few attempts at building reinforcement learning agents for Pokémon, most notably the IEEE 2017 CIG (Computational Intelligence for Games) conference presented a Pokémon Showdown competition [10] with a baseline Q-Learning agent with poor results against other classic algorithms. The authors tried a singer layer perceptron and a multiplayer perceptron to estimate the value function, but little detail is described on the paper.

A team of students from Stanford's CS221 class [11] also developed a Reinforcement Learning player using Expectimax and an evaluation function trained on data from 20,000 Showdown games using TD-Lambda learning. The results were promising but still very far from the top players, the agent achieved an ELO score of 1340 V.S. top ranked players at 1800. The authors acknowledge training on expert players may have negative effects when battling novice players, as top players follow strategies optimized against players of similar level. Another interesting aspect of their work was that the agents were evaluated online against real showdown players. They also used a Showdown open source client [12] from a Showdown forum member that could be adapted for this project.

Outside of Pokémon there has been significant research on self-learning agents

from the Google DeepMind team, with published superhuman performance on Go [4], Shogi and Chess. Their work outlines a simplified implementation of Monte Carlo Tree Search (MCTS) and a single neural network that outputs state value (chance of the player winning at that state) and move probabilities in a single pass. This approach would most likely be the starting point of the project given the success of MCTS across many other games such as in AGI's 2016 game competitions [13].

1.3 Stakeholders

1.3.1 Student and co-directors

The main people involved in this project are the student and bachelor thesis co-directors. For both parties this is the first time they do research on self-play for RL agents and the main interest is in learning as much as possible in the subject. Both of the group's interests is in learning the basics and the challenges in training a RL agent to play a game without previous information.

1.3.2 Beneficiaries

Competitive Pokémon players

If the RL agent reaches a strong level of play, competitive Pokémon players could use it for training and improving themselves. In other domains with strong AI agents such as chess, players are known to practice against the AIs in preparation for ranked matches. In the case of Go, AlphagoZero has been used by the world's best players to learn more about the game itself due to the agent's unique approach to playing.

1.4 Scope

1.4.1 Objectives

1.4.2 Primary Objective

The main objective of the project is to develop a RL agent trained through self-play that succesfully plays a simplified version of Pokémon battles in the PokemonShowdown environment. This restricted version will include at least 2vs2 Pokémons chosen at random from a small subset of available choices.

The project will initially set up the learning environment 1.4.2 and then explore several baseline algorithms such as a greedy approach and basic tree search. Afterwards a sample RL agent will be implemented using a small neural network and TD-learning to act as a baseline. The rest of the time will be spent implementing MCTS enhanced by the evaluation and policy functions, and training the functions through self-play following a similar approach to AlphaZero. The end results will be benchmarked against each other and an analysis on the agent behavior at different stages of the learning will be carried out.

1.4.3 Secondary Objective

The secondary objective will be to train the agent to play the full game (6 Pokémons per team) and evaluate it's performance online against real players in the PokemonShowdown platform. The aim will be to achieve the level of an amateur player as measured by ELO score.

1.4.4 Obstacles

Computational Power

The biggest risk in this project is the lack of computational power, in this field the speed of iteration is directly correlated with the resources available. A team with thousands of Graphic Processing Units will try more experiments and will be able

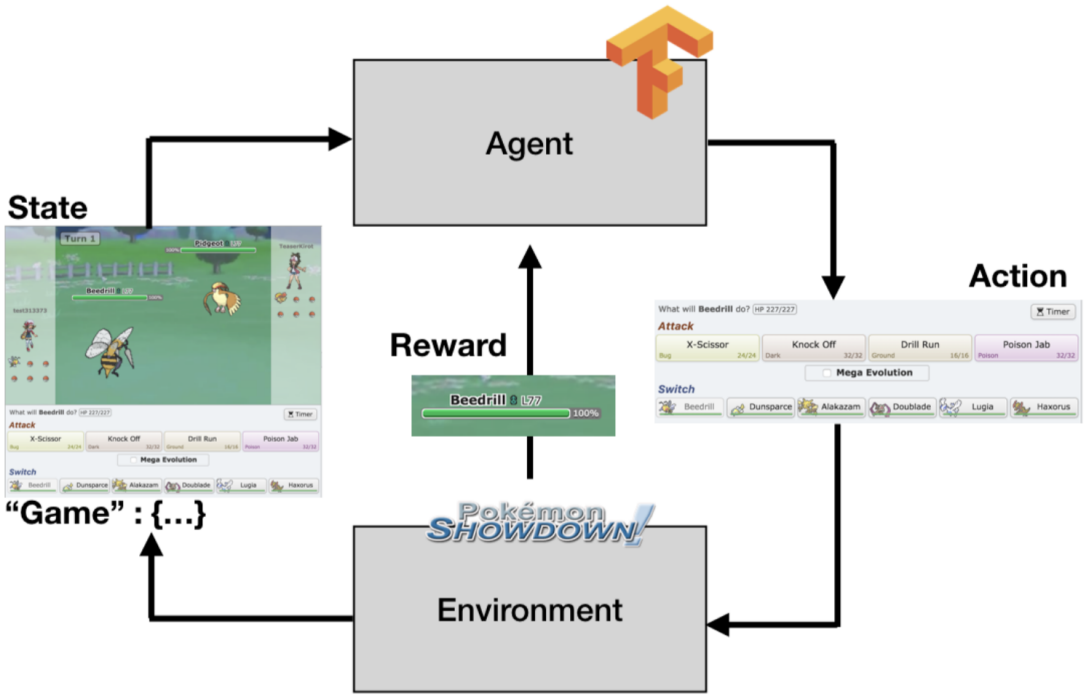

Figure 1-2: RL Agent architecture for Showdown

to train the neural networks further to achieve better results. A lack of GPUs will limit the quality and feasibility of the project and will make it harder to achieve the secondary objectives. For this project there is a small number of GPUs available for research which should be sufficient to get some initial results but perhaps not enough to train the agent to play the full game.

Domain Complexity

The game of Pokémon has many hidden properties that could prove to be too complex to be treated with Reinforcement Learning with the available computational resources. There are no previous attempts in the literature to tackle a problem with these characteristics with self-play.

1.5 Methodology

Due to the complexity of the project, an Agile/Lean approach will be highly beneficial. A combination of goal-oriented short development cycles and a frequent cadence of meetings will be key in ensuring the success of the project.

1.5.1 Short development cycle

The work will be carried out in short iteration cycles with goals set on a weekly basis. This will help reduce the risk of hitting complex problems and trying to solve them for too long. This will also serve as a good tool to measure progress, if no value is delivered after two iterations this will be a warning sign that the project is not on track. This will also keep focus on the most valuable tasks at any given week which should help in the development of the project. These objectives will be set in weekly meetings with the co-directors and will steer the project in the right direction.

1.5.2 Frequent measurable experiments

A key method of work in this field of research is that of frequent measurable experiments. Throughout the development of this project, a stable experimenting platform will be used to run the agent against a set of set environments and success metrics and performance will be compared with the baseline agents. This will be one of the key methods to keep track of progress towards the goal.

1.5.3 Monitoring Tools

A Git repository hosted on Github will be the main tracking tool for the project. All work will live on a source repository for complete transparency and progress tracking. The Github project includes an issue tracking system that will be used to document the progress and key questions during the project. There will also be regular bi-weekly progress updates over email, sent to all relevant stakeholders.

Chapter 2

Project Planning

2.1 Schedule

2.1.1 Estimated project duration

2.1.2 Considerations

The estimated project duration is of about 4 months. The project starts on Thursday 22nd February, 2018 and the deadline is on Sunday 24th June, 2018.

The following project plan is subject to change as the project evolves, the use of agile methodologies and experiment results may alter the original plan.

2.2 Task descriptions

2.2.1 Acquire background in Deep Learning

Since September 2017 I have been self-studying the field of Deep Learning (DL), specifically deep neural networks and convolutional neural networks. I completed the first four courses of the Deep Learning Coursera course by Andrew Ng (Stanford, Google and Baidu) [14], the courses serve as an in-depth introduction from first principles of the field. As part of the course I did lectures and assignments on deep neural networks, hyperparameter tuning, deep convolutional networks and deep learn-

ing project management. Concurrently I started following the Deep Neural networks book [15] for a detailed view on the topics along with a more formal mathematical formulation of the subject. The main resource needed for this project was time, and access to the (paid) Coursera course with instructor evaluation and assignment environment.

2.2.2 Acquire background in Deep Reinforcement Learning

With the previous knowledge in Deep Learning, an understanding of basic Reinforcement Learning (RL) was needed to proceed with the project. I began this task the month before starting the thesis by reading the reference book in the field by Sutton and Barto [16] and completing the first chapters to get an overview of the subject. The rest of the time was spent understanding the seminal paper in Deep Reinforcement learning [2] introducing DeepQ Learning, with the objective of replicating the results with an open source implementation. I set up a popular open source reinforcement learning library and read and ran the simple examples replicating parts of the DeepMind paper and tweaking the hyperparameters to get acceptable results. After these initial projects I focused my time on understanding the latest self-learning approaches with the main papers in the subject and running a baseline implementation of AlphaZero [5]. The requirements for this part were again time and access to the learning resources (book and articles) but also access to a moderately powerful machine to run the deep learning examples.

2.2.3 Setting up research environment

The correct development of this project depends on using the computational resources available at the imatge department [17] at ETSETB. For the duration of this project a number of GPUs and machines are available for development and experimentation. This environment is something that I have never worked with and I will need to get acquainted with the main methods of operation and best practices. This will include understanding the resources available, the queuing system (srun) and any

other details that may be unknown at this moment. During this phase I intend to run the examples I explored in the previous two tasks in the machines at the imatge cluster. This task will require human resources to read and understand the cluster environment as well as access to at least one machine with Graphic Processing Unit resources.

2.2.4 Learning OpenAI Gym and Universe

As described in the project specification, the PokémonShowdown [9] game will be ported to an OpenAI Universe environment. The OpenAI Universe environment is an open source framework that defines RL environments via a set of standard specifications that many libraries and platforms can consume to train agents via different environments. This task's objectives are to understand how the OpenAI gym (simpler version of the Universe) and Universe work, porting a simple game to the platform and training an agent with a basic algorithm such as DeepQ Learning. Besides the human resources to understand the OpenAI platform and write code, a machine in the imatge clusters will be needed to test the implementation.

2.2.5 Porting Pokémon to OpenAI Universe

The objective of this task will be to port the PokémonShowdown game to an OpenAI Universe environment. This is a requirement because the rest of the project will leverage the standard Universe format to experiment with existing RL baselines. The main work of this part will be packaging the existing PokémonShowdown battle server into a Docker container so that it can be executed without external dependencies in the imatge cluster. After the packaging, code will need to be written to interface the Showdown server APIs with the Universe format, this is a straightforward task however requires extensive development and testing work. The main resources for this task will be human resources to do all the dockerization and Universe API mapping, the work will also require access to imatge's cluster resources and support from an engineer in the project.

2.2.6 Setting up experimentation platform

With the Universe environment set up, a key part of the project will need to be developed: the experimentation platform. The goal of this platform is to evaluate different versions of the Reinforcement Learning agent and run it against different baselines and versions of itself. This will serve to obtain the results of the thesis as well as guide development and iterations of the project. The task will require human resources to develop and test the code as well as access to one imatge machine to test the execution in practice.

2.2.7 Implementing baselines

With the experimentation platform set up, it will then be time to implement the baseline agent implementations that will be used to evaluate the agent. The algorithms that will be considered will be the following: Random, Greedy, BFS, Minimax, DQL, TD-Learning and TD-Learning + Minimax. These baseline agents will be used by the experimentation platform to do a first evaluation of the Showdown Universe environment. This task will require human resources for the development of the code as well as access to one of imatge's machines

2.2.8 Replicating AlphaZero

The continuation of the work will be centered in replicating the AlphaZero paper by leveraging an open source implementation. The AlphaZero implementation in the Showdown environment will be the main part of the thesis ad this will take considerable time. This task will require human resources for the development of the code as well as access to one of imatge's machines.

2.2.9 Hyperparameter tuning and analysis

With an implementation of the AlphaZero algorithm, the final task will consist in tuning the agent hyperparameters to successfully train the agent in playing PokémonShowdown battles. This will be carried as long as possible as it is a very manual

task with many trial and error attempts. The task will finalize by analyzing the resulting agent behaviors and writing a report on it's performance against the baseline agents.This task will require human resources for the development of the code as well as access to several imatge machines.

2.3 Estimated time

Table 2.1: Estimated project time

Task	Estimated duration (h)
Acquiring background in Deep Learning	70h
Acquiring background in Deep Reinforcement Learning	15h
Setting up research environment	10h
Learning OpenAI Gym and Universe	15h
Porting PokémonShowdown to OpenAI Universe	40h
Setting up experimentation platform	20h
Implementing baselines	90h
Replicating AlphaZero	90h
Hyperparameter tuning and analysis	90h
Total	440h

2.4 Gantt chart

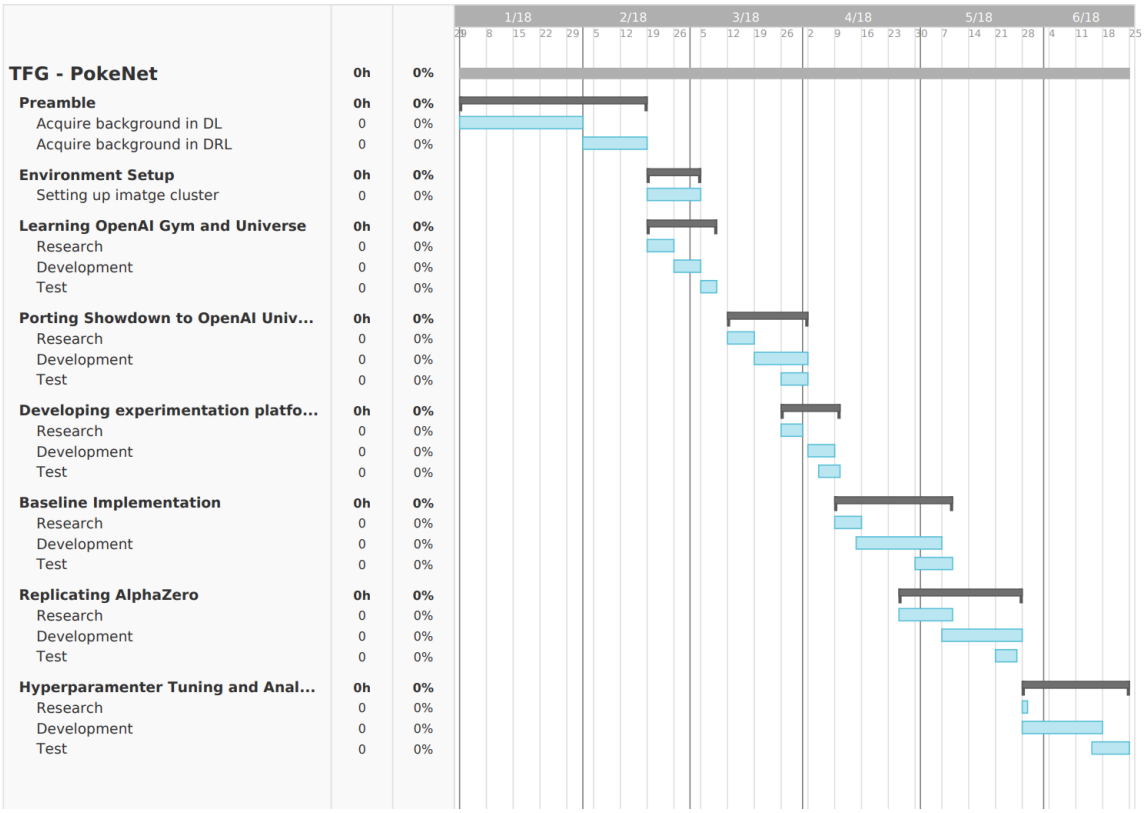

Figure 2-1: Project task Gantt chart

2.5 Alternatives and action plan

As stated in the project context and scope, using an agile methodology will help us dynamically adapt this plan. If timing is different than planned, the organization will be modified to adapt for faster or slower development times and negative experiment results. Bi-weekly meetings are set in place with the thesis co-directors and this will help detect problems early during the development.

With an average dedication of 30h and 16 weeks of work, 480h should be enough to accomplish the project.

Following, a list of potential problems and alternatives:

2.5.1 Hard to replicate algorithms

It is generally hard to replicate published papers in Deep Learning, this is a problem in this field of study. Many of the baselines are based on previously published work, there is risk that it is hard to reproduce the work. The main method to mitigate this risk will be to use Open source implementations as often as possible to isolate programming errors from replicating results. This will allow focus on adapting existing algorithms to the problem at hand - there are several implementations identified that fit well for this project and with a few alternatives to choose.

2.5.2 State representation being too complex

The full game of Pokémon is highly complex and a full state representation is highly dimensional (50k features). This may slow down training and could limit progress greatly. This risk can be mitigated by representing the state with a subset of features, this will effectively enable the project to go forward but may lead to worse results.

2.5.3 Computational power insufficient

Deep Reinforcement Learning agents need strong computational resources, without a number of powerful GPUs it is simply impossible to train the agents. If the resources available in imatge's clusters prove insufficient, the project could get stalled. A possible remediation could be to use other research group's resources or to get some cloud computing credits from industry collaborations. For short periods of time (such as when presenting final results), this alternative would effectively mitigate the risk.

2.5.4 Computational power unavailable

The resources at imatge's cluster may be unavailable due to high use by in-house researchers, this may slow down the development of the project as no code can be ran. A reserved machine within the cluster could be requested just for carrying on the project, and this would solve the problem.

2.5.5 Libraries unavailable in cluster

This project leverages several open source libraries and complex frameworks such as Docker. These platforms may prove challenging to install in imatge's clusters, if it is not possible an alternative will need to be found to run the Showdown environment. It could also be possible to host the server externally (in Amazon Web services or Google Cloud) and connect to the cluster via the internet. While this would solve the problem it would also slow down the learning.

Chapter 3

Budget and sustainability

3.1 Project budget

In this document you will find an estimation of the project costs, including hardware, software and other indirect costs.

3.1.1 Hardware budget

In order to train the self-playing agent described in the previous section, a set of hardware will be required - the main cost will be that of the machine and graphics card used for training. The costs of the machine used for training are an approximation of the hardware required - the specifications are similar to those used during the study and hosted in the imatge department cluster.

Product	Price	Units	Useful Life
Macbook Pro 13" 2,3 GHz	1505,59 €	1	5 years
NVIDIA GTX 1080Ti	849,53 €	1	3 years
PcCom StartUp1 Intel i5-7400 / 16GB / 120GB SSD	662,45 €	1	5 years
Total	3017,57 €		

Table 3.1: Hardware Budget

3.1.2 Software budget

This project is built entirely using free and open source software so the cost is zero.

Product	Price	Units	Useful Life
ShareLaTeX	0 €	1	N/A
Vim	0 €	1	N/A
Git	0 €	1	N/A
Github	0 €	1	N/A
Google Docs	0 €	1	N/A
TeamGantt	0 €	1	N/A
TensorFlow	0 €	1	N/A
Tensorforce	0 €	1	N/A
OpenAI Universe	0 €	1	N/A
Pok{{́e}}mon Showdown server	0 €	1	N/A
Total	0 €		

Table 3.2: Software Budget

3.1.3 Human resources budget

The project is being developed by a single person. This person will play the roles of the project manager, software engineer and tester. Each role is split in the total 440h.

Product	Hours	€/ hour	Total
Project Manager	50	50	2.500 €
Software Engineer	300	35	10.500 €
Tester	90	30	2.700 €
Total	440		15.700 €

Table 3.3: Human resources budget

Work time breakdown for the three roles is as follows:

Task	Estimated duration (h)	Dedication (h)		
		Project Manager	Software Engineer	Tester
Acquiring background in DL	70h	15	30	25
Acquiring background in DRL	15h	5	5	5
Setting up research environment	10h	0	10	0
Learning OpenAI Gym and Universe	10h	0	10	0
Porting Showdown to OpenAI Universe	40h	0	25	15
Setting up experimentation platform	20h	5	10	5
Implementing baselines	90h	5	75	10
Replicating AlphaZero	90h	0	75	15
Hyperparameter tuning and analysis	95h	20	60	15
Total	435h	50	300	90

Table 3.4: Work time breakdown by role

3.1.4 Unexpected costs

During the project development, some deviation might happen, so an extra contingency budget is planned in case extra work is required

Product	Hours	€/ hour	Total
Project Manager	15	50	700 €
Software Engineer	15	35	525 €
Tester	10	30	300 €
Total	440		1.525 €

Table 3.5: Contingency plan human resources budget

3.1.5 Indirect costs

This section includes all costs not accounted for in the previous sections. Note this includes significant energy costs as the project will require continuous use of a desktop

workstation/server to train and develop the agent.

Product	Price	Units	Cost
Electricity	0,07 €/ hour	3000 kWh	210 €
Internet	35 €\month	4 months	140 €
Total			350 €

Table 3.6: Indirect costs

3.1.6 Total budget

The total project cost is calculated by combining all the previous budgets together. A contingency cost of 5% of the total budget is added to account for any extra expenses incurred during the project.

Concept	Estimated cost
Hardware	3.017,57 €
Software	0 €
Human resources	15.700 €
Unexpected costs	1.525 €
Indirect costs	350 €
Subtotal	20.592,57 €
Contingency (5%)	1.029,63 €
Total	21.662,20 €

Table 3.7: Total budget

3.1.7 Breakdown per task

Task	Estimated Cost
Acquiring background in DL	2550 €
Acquiring background in DRL	575 €
Setting up research environment	350 €
Learning OpenAI Gym and Universe	350 €
Porting Showdown to OpenAI Universe	1325 €
Setting up experimentation platform	750 €
Implementing baselines	3325 €
Replicating AlphaZero	3075 €
Hyperparameter tuning and analysis	3550 €
Total	15850 €

Table 3.8: Cost estimation breakdown by task

3.2 Budget Monitoring

In order to keep track of the budget, an checkpoint will be made at the end of each task with the real number of hours, this will be compared to the estimated budget. This will give early visibility to any discrepancies with the original plan and will help adjust the budget accordingly

3.3 Sustainability and social commitment

The sustainability of the project is analyzed accross three dimensions: environmental, economical and social. The analysis is carried out by using the sustainability matrix proposed in the course. Overall the results are positive with a sustainability range of 69/90, no particular area is highly unsustainable.

	Project Development	Exploitation	Risks
Environmental	Consumption Design	Ecological footprint	Environmental risks
	9/10	18/20	-2/-20
Economic	Project Bill	Viability plan	Economic risks
	6/10	14/20	-2/-20
Social	Personal impact	Social impact	Social risks
	9/10	17/20	0/-20
Sustainability	24/30	49/60	-4/-60
Range	69/90		

Table 3.9: Project sustainability matrix

3.3.1 Environmental dimension

The main resource used in this project is the electrical energy required to power the machines used in training the self-learning agent. While the energy required is not excessive, it is well above the normal consumption of a laptop or desktop PC. This resource is not reusable, but all the equipment needed for the project can be re-purposed for future research or consumer use.

The project also plans to reuse the computational resources (GPUs) at the ETSETB Image Processing group cluster. This will reduce waste as other researchers can share the same infrastructure for multiple research projects, and specifically to maximize the resources when others are not using them.

3.3.2 Economical dimension

The project has a well defined budget across all resources needed. The cost is on-par with similar experiments in the literature (using 1 GPU for training), but severely

below state of the art implementations ($>$ 100k €) due to reduced scope.

3.3.3 Social dimension

The execution of the project has no ethical or professional implications as it is an exploratory work for building a better AI to play a popular computer game. At a personal level the work is very fulfilling as it is an opportunity to dig deeper in a very hot topic in technology. If good results are published this could help develop my professional career in the field of Deep Learning as others would be interested in my skill-set.

The work could have some positive impact in the broader research community - it will set new baselines for using reinforcement learning and self play in a new domain.

Chapter 4

Game Analysis

In the Pokémon game, a player's character ("trainer") travels across a region capturing creatures called Pokémon and trains them for battle. There are hundreds of species of Pokémon, and many Pokémon are able to evolve into other Pokémon, which typically makes them larger and stronger. As they catch, train and evolve Pokémon, the trainer prepares their team for the primary game mechanic of the game: battles. The contents of this section aim to give a comprehensive summary of the characteristics of the battles.

4.1 Mechanics

4.1.1 Battles

Battles feature two teams, of up to six Pokémon each, battling each other. One Pokémon per team is active in the battle at a time, however 2 vs and 1 vs many modes exist in the latest generation games. Players can select up to six of their Pokémon to form their team; while team selection can play an important strategic role in gameplay, this is out of scope for this work.

Figure 4-1: Gameplay screen: Opposing Pokemon, both indicating full health (HP).

Winning condition

In order to win a battle, a player must defeat all the opponent's Pokémon. All Pokémon have a property called Health Points (HP), which is reduced each time that Pokémon takes damage from attacks. When the Pokémon's HP value is reduced to zero, the Pokémon "faints" and is unable to battle. A player wins the battle when all Pokémon on the opposing team have fainted.

Turns

Pokémon uses a simultaneous execution-based turn system. Rather than alternating turns and choices from player to player, both players submit their decision, and the game resolves the decisions simultaneously.

In a single turn, a player can choose to either attack using one of their active Pokémon's moves, or switch the Pokémon for another Pokémon in their team.

Once both players choose a move, a priority system is used to resolve the events of the turn. If either side chooses to switch their Pokémon, that action is performed before the attacks. Then, certain moves with special priority are executed. If both players chose moves without special priority, the Pokémon that moves first is the Pokémon with the higher "speed" statistic.

Figure 4-2: Pokemon Types

4.1.2 Pokémons

In current generations of the Pokémon game, there are over 800 distinct species of Pokémon which a player can choose to use in battle. In addition, the training, nature and independent attributes of each Pokémon within a species can differ, leading to a high degree of variability in the composition of Pokémon teams.

Types

A Pokémon's type determines the strengths and weaknesses of the Pokémon and its moves. Pokémon may have one or two types, and there is a total of 18 types (Figure 4-2). When an attack is used on a Pokémon, the damage inflicted is impacted by a multiplier based on the Pokémon's type; this is described in detail in section 4.1.3.

Some types also have properties which are unrelated to the damage chart. For example, Electric-type Pokémon are immune to being paralyzed, Water-type are immune to being burned, etc.

Stats

Each Pokémon has six individual statistics ("stats"), in addition to two in-battle stats that are the same for all Pokémon and can be modified during battle.

- **Health points (HP)**: how much damage a Pokémon can receive before fainting. A Pokémon faints when its HP reaches zero.
- **Attack**: partially determines how much damage a Pokémon inflicts when using a physical attack.
- **Defence**: partially determines how much damage a Pokémon receives when it is hit by a physical move.
- **Special Attack**: determines the strength of a Pokémon's special attacks.
- **Special Defence**: determines the resilience of a Pokémon to special attacks.
- **Speed**: determines the order that a Pokémon moves in battle; a Pokémon with higher speed will execute its move before a Pokémon with lower speed.

In-battle stats of **Accuracy** and **Evasiveness** are not innate Pokémon traits, but can be modified during battle. A Pokémon with higher Accuracy has a higher probability of its move hitting its opponent; high Evasiveness makes an opponent's move more likely to miss.

Moves

Pokémon are able to learn a variety of moves depending on their species, and can learn up to four moves at a time. Moves can have various effects on the opponent, the user, or the gameplay environment, such as inflicting damage, inducing status changes, restoring health, or changing the weather.

4.1.3 Moves

Moves can be defined as **physical moves**, **special moves**, or **status moves**, and there are 728 unique moves in total. Physical moves are those that would make physical contact (e.g., punches, bites), while special moves typically show control of the elements (e.g., breathing fire, using the wind). Physical moves and special moves inflict damage, reducing the opponents HP, based on the type and stats of the Pokémon and the move.

Status moves do not deal direct damage, but can inflict status changes either on the user or the opponent. For example, Poison inflicts damage on the opponent every turn, while Sleep causes the opponent to be unable to use its moves for a random duration of turns.

Stats

Each move has four stats:

- **Power Points (PP)**: how many times a Pokémon can use that move. Once a move's PP is depleted, a Pokémon may no longer use that move.
- **Accuracy**: probability of move hitting the target.
- **Power**: amount of damage dealt by the attack. Very powerful moves are often offset by below-average accuracy and/or low PP.
- **Type**: move type, used in damage calculation.

In addition, some moves may have negative effects on the user, such as inflicting recoil damage, or forcing the player to wait an extra turn after use. There are also moves, such as Splash, which have no effect. Finally, some moves produce delayed rewards long after the user has been defeated; "Stealth Rock" creates an "entry hazard", causing damage to a Pokémon whenever they switch into battle.

Accuracy

The probability of a move successfully hitting the opponent Pokémon is determined by the formula:

$$T = Accuracy_{\text{move}} * Accuracy_{\text{user}} * Evasion_{\text{target}} \tag{4.1}$$

Where:

T is the threshold value, between 1 and 255, that determines whether the move will hit.

$\text{Accuracy}_{\text{move}}$ is the move's accuracy, ranging from 0 to 255.

$\text{Accuracy}_{\text{user}}$ is the accuracy multiplier of the user.

$\text{Evasion}_{\text{target}}$ is the evasion multiplier of the target.

The game calculates a random number r between 0 and 255, and compares it to T to determine if the move will hit. If r $<$ T, the move will hit the opponent Pokémon.

Damage

The damage inflicted on the opponent Pokémon by a move is calculated according to the following formula:

$$Damage = (\frac{(\frac{2*Level}{5} + 2) * Power * A/D}{50} + 2) * Modifier \tag{4.2}$$

Where:

Level = level of the attacking Pokémon

Power = the effective power of the move being used

A = the Attack stat of the attacking Pokémon if a physical move, or the Special Attack stat if a special move

D = the Defence stat of the attacking Pokémon if a physical move, or the Special Defence stat if a special move

The modifier is calculated as:

$$Modifier = Weather * random * STAB * Type * Other \tag{4.3}$$

Where:

Weather = Environment multiplier, depends on weather and Pokémon type: If the game environment is harsh sunlight, a Water-type move will be 0.5 and a Fire-type move will be 1.5 If the game environment is rain, a water-type move will be 1.5 and a fire-type move will be 0.5. Otherwise, weather = 1.

Random = a random factor, [0.85, 1.00] that introduces an element of luck into the damage inflicted by a move.

STAB = If a Pokémon has the same type as the move it uses, it will receive a Same-Type Attack Bonus (STAB), which increases the damage of the move by 1.5.

Type = Each move is classified as one of the 18 Pokémon types, and a move's damage is dependent on how effective a move is on the target Pokémon's type. Type effectiveness can range from 0 (ineffective) to 4 (super effective).

Finally, other components may influence the modifier, such as a status change (e.g., burn) or the chance event of a Critical Hit, which occurs with random chance and roughly doubles the damage of the attack.

Type

The type effect is calculated based on the type of the move and the type(s) of the target Pokémon (Figure 4-3. Pokémon receive double damage from types they are weak to (2x), and half damage from types they are resistant to (0.5x), according to the type bonus matrix below. A type multiplier of 1 indicates a move that is normally effective against the target; 0 is ineffective, 0.5 is not very effective, and 2 is super effective.

If a Pokémon has two types, these weaknesses and resistances are combined: the multiplier is calculated for each type and and multiplied together, potentially resulting in 0.25x or 4x multipliers.

Figure 4-3: Type damage modifier matrix.

4.1.4 Human Player Strategies

Several battle strategies have emerged through the years from competitive players [18]. It is not the purpose of this project to explore them in detail, however it is important to understand how humans play as this will serve as a good reference to recognize intelligent play:

- **"Jobs"**: the move set of each Pokémon determines it's "Job" in the battle. Trainers craft their teams around said jobs and they perform specific duties. Examples include "Sweepers", attack/special attack focused Pokémons whose job is to take down many opposing Pokémon per battle. "Wallers" on the other hand focus on their Defence and Special Defence and are meant to counter "Sweepers" with moves that boost their defense. Finally many Pokémons are built around a highly effective attack, for example "Spikers" whose focus is to to introduce many entry hazards in the battle field.

- **Move Combinations**: a big strategic advantage is achieved when combining the right moves together. One such example is "Baton pass" combined with

moves that boost the attack/special attack stats. The Pokémon begins by exclusively boosting itself, then uses "Baton Pass" which changes the current Pokémon for another in the team all while passing the attack boosts - this is highly effective when changing into a "Waller" which will have really strong stats and will throw the opponent off guard.

4.2 Challenges

4.2.1 Imperfect Information

In Pokémon the opponent's team is hidden by default and is only partially revealed as Pokémons come into the field and moves are used. Opposing Pokémon's stats are never revealed, except for HP and level.

4.2.2 Turn Atomicity

Pokémon is modeled as a **Normal-form** game, each turn both players submit a move that is evaluated simultaneously by the engine. This is opposed to **Extensive-form** games, where games can be described in tree form, composed of sequences of moves and game states. This can result in scenarios where players execute moves in environments different than when their action was selected.

4.2.3 Categorical Dimensions

Pokémons can have many different status effects that are difficult to quantify and compare. For example, a Pokémon could be burned and paralyzed or any of the several other status effects; it is not straightforward to measure the compound effects that a combinations of status effects many have.

4.2.4 Simulation Cost

In current available Open Source implementations of the Pokémon Battles engine, turns can take 5 to 40ms depending on the complexity of the games. This elevated cost of simulation limits the number of states that can be explored by agents simulating many turns such as Monte Carlo Tree Search (MCTS) or any other type of tree search.

4.2.5 Stochasticity

Pokémon is not a deterministic game and incorporates several randomness elements in the game, as described in the previous sections. Attack damage, attack success and status alterations are all examples of game elements that have a chance element. This is an added challenge as agents can't perfectly model state transitions.

4.2.6 Branching Factor

Ignoring the imperfect information and stochasticity, the branching factor of the game is usually between four and nine (four moves and 5 possible Pokémon changes). However, the maximum branching factor is 25, in case the "Baton Pass" move is used, a Pokémon and that Pokémon's move needs to be selected.

Chapter 5

Self Play

In Reinforcement Learning (RL), a method that has been used successfully for training agents to play complex games is self-play. Self play was introduced in early studies of RL agents for Backgammon, and it consists of an agent playing many games against itself and using the "reward" signals at the end to improve. The method has two advantages that are very relevant to the game of Pokémon:

- Agents can be trained without data-sets: the publicly available datasets of Pokémon expert games is minimal. Self-play agents can start learning from scratch starting from random play [5].

- Agents learn complex domains better by iteratively increasing the difficulty: this is particularly attractive in a really complex game such as Pokémon. This property has been applied in the past in complex 3D world simulated physics competitive problems [19].

5.1 Reinforcement Learning

Reinforcement Learning is a method by which agents learn how to interact with an environment through actions in order to maximize rewards. In this type of problems, the environment is usually modeled after a Markov Decision Process (MDP).

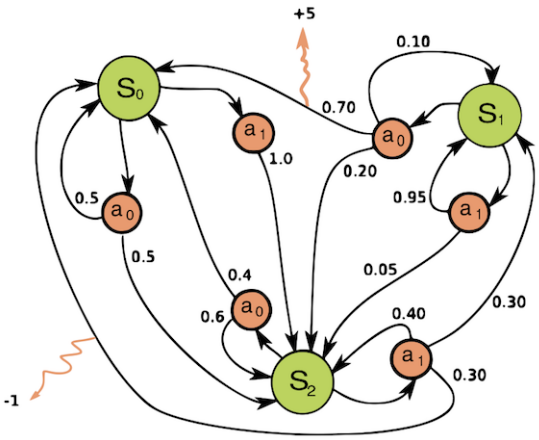

Figure 5-1: Markov Decision Process.

Markov Decision Processes

MDPs are described by a 5 tuple (A, A, P_a, R_a, γ)

Where:

S is a finite set of environment states.

A is a finite set of agent actions.

P_a (s,s') is the probability that action a in state s at time t will lead to state s' at time t+1.

R_a (s,s') is the reward received after transitioning from state s to state s', due to action a.

$\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards. $\gamma^0 R_1 + \gamma^1 R_2 + \gamma^2 R_3 + ...$

The method also has it's challenges, such as the explore-exploit dilemma (choosing non-optimal actions for better opportunities in the future) and the credit assignment problem (which action to attribute a reward to).

5.1.1 Deep Reinforcement Learning

Deep Reinforcement Learning combines Neural Networks as function approximators and Reinforcement Learning algorithms. Neural Networks are really powerful in RL

	Information	Environment	Game Type
AlphaZero (Go, Chess)	Perfect	Deterministic	Extended-form
Pokémon Battles	Imperfect	Stochastic	Normal-form

Table 5.1: Go/Chess problem differences with Pokémon Battles

problems, as states are often highly complex such as in Pokémon and evaluating with hand-crafting features is prohibitive.

AlphaZero

AlphaZero [20] is a self-play deep reinforcement algorithm designed to play generic deterministic extended-form games. AlphaZero makes use of a Neural Network estimating a policy network (moves most likely to win the game) and a value network (win probability for a given state). The approximations are used to intelligently direct a Monte Carlo Tree Search, which explores a game tree, represented by states and player actions.

5.2 Challenges with self-play RL in Pokémon Battles

By exploring the problem of Pokémon Battles in the scope of self-play, we uncovered several challenges that are current open-ended areas of research. The first realization was that AlphaZero can not be used as an algorithm of choice due to the types of problems it is designed to solve (Table 5.2).

One of the main issues is with with the normal-form game type [21]. Normal form games can't be modeled as a sequence of moves executed by players one after the other. This is one of the main properties exploited by MCTS when traversing the tree. There may be work-arounds this challenge such as representing state transitions as pairs of movements of both players, but with imperfect information the branching factor explodes to 320,000 (number of possible pokemons (800) * number of possible moves (400)). Since the opponent's Pokémons and their moves are not known, the possibilities are intractably large.

5.2.1 Stochasticity

Firstly, AlphaZero's tree search exploration is based on discrete actions that can be mapped to exact environment changes. A single move in chess or Go, leads to a predictable board position. In Pokémon, however, as described in section 4, actions have a probability distribution of possible outcomes, and this can't be modeled in a classic search tree.

There are several attempts in the literature to overcome this. One group of researchers has developed an Upper Confidence Stochastic Game Algorithm [22], that estimates transition probabilities with empirical frequencies and builds a confidence region for each transition probability. The agent then chooses an optimistic model and finds the optimal policy to play with.

Another such approach is to model the moves as Mixed Strategies [23], probability distributions that are accessible through a parameterized function of finite dimensions. The researchers then adapt these mixed strategies with a fixed set of dimensions to be used in classic reinforcement learning algorithms.

5.2.2 Imperfect Information in game state

The problem of imperfect information is a key one, as AlphaZero makes use of a perfect information state to compute it's value and move probabilities. However such methods lack convergence guarantees for imperfect information states, and they can't be blindly used to treat a problem like Pokémon Battles.

One possible way to tackle this problem is to use the Neural Fictitious Self Play (NFSP) [24] algorithm, which conceptually averages the player's behaviours and has good results under uncertainty conditions. In NFSP, the algorithm stores two classes of memory and has two separate neural networks. The first stores games against other agents, and learns to predict expected value. The second stores the agent's actions, and tries to predict it's average behavior. The algorithm then carefully samples between the greedy, highest-value actions and the agent's average moves. This methods achieves great results in Leduc Poker, reaching close to Nash-equilibrium.

Chapter 6

Self Play Frameworks

In the development of self-play Reinforcement Learning agents, there are many challenges that researchers have to overcome to get started and obtain meaningful results. One of the highest barriers to entry is the availability of tooling to conduct research: self-play algorithms are often complex and make use of concurrency and multiprocessing, therefore researchers can spend a significant amount of time engineering solutions that could be best spent on the actual research. The barrier of entry for beginners is high and there are very few tools available to get started with simple problems.

As part of this project, several libraries were evaluated, with the goal being to find tools that filled two gaps:

- Ease to incorporate new and complex environments.
- Off the shelf support for self-play with baseline implementations.

One such library was found, and a simple multi-agent self-play environment was implemented with the objective to introduce early researchers to the field. All other leading RL libraries did not provide support for self-play capabilities off the shelf, libraries surveyed included TensorForce, keras-rl, and OpenAI Gym.

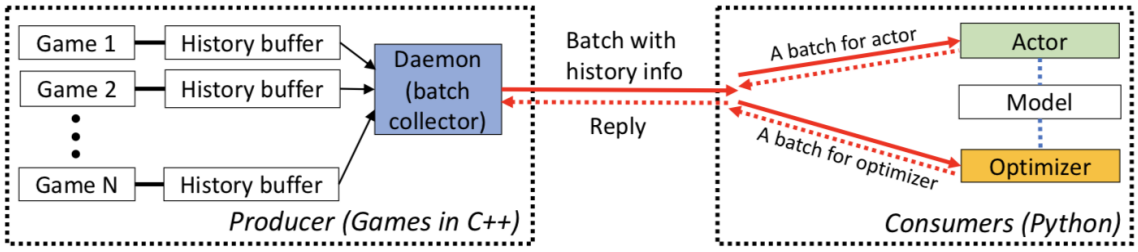

Figure 6-1: Overview of ELF architecture. The producer-consumer topology can be configured through code, enabling many different data models.

6.1 ELF

The Extensive, Lightweight and Flexible (ELF) [25] Research Platform is a tool developed by Facebook Inc. to conduct fundamental Reinforcement Learning Research. The platform includes a hybrid Python/C++ framework with configurable topology of environment-actor communications used in training. This enables researchers to implement one-to-one, many-to-one and one-to-many configurations (Fig 6-1). This includes all ranges of algorithms, from single threaded Deep-Q learning to multi-threaded Asynchronous Actor Critic [20], an algorithm using multiple copies of an agent in parallel updating a global neural network.

ELF has also been used to implement the first open source version of AlphaZero, OpenGo [26], and provides significant code to be reused for self-play. The framework comes with several research environments available, including a custom made simplified version of a Real Time Strategy game similar to Starcraft (RTS), and a version of Go.

The framework comes with a deep reinforcement learning python library built on top of the PyTorch deep learning framework and includes several baseline algorithms that can be adapted to each game. In order to do this, the neural network used in each case needs to be modified to fit the targeted game. As of this date ELF has support for Deep-Q learning, Asynchronous Actor Critic and Policy Gradient methods.

One of the strengths of ELF is it's performance; the core of the engine is developed using C++ as opposed to python multiprocessing. The tool claims a 3x speedup on several tasks including the Arcade learning Environment, a combination of classic

arcade games usually used to benchmark frameworks. According to their observations, python multiprocessing with OpenAI Gym environments is even slower, due to heavy communication of game frames between processes.

While very powerful and extendable, ELF comes with little to no documentation and it's hybrid python/c++ codebase can be challenging for many looking to get started. The available games are highly complex to a beginner looking to get started doing self play in reinforcement learning, which is shown by the lack of external contributions to the project. Since inception (2017) the framework has seen no new games added by external contributors, however several have asked for directions or requested other games to be added. While the framework includes a python wrapper for the core functionality, it lacks proper support as many of the core features are missing.

6.1.1 Proof of Concept

Due to the complexity of existing games implemented in ELF, and the challenge of no external contributions to the framework, work was done to implement a simple multi-agent game with self-play capability. The chosen game was multiplayer snake, as recommended by the OpenAI Requests for Research 2.0 [27] as a good starter problem for self-play reinforcement learning. This game was integrated into the ELF framework, tested end-to end and it's performance measured. In the future we hope to contribute the game to the main ELF repository to help open self-play to a larger audience.

Game

The game is identical to classic snake, with the added twist of multiple snakes, each controlled by a different player, in the same 2D grid. Snakes start with unit length and grow by eating fruit. A snake dies if it attempts to exit the grid or if it collides with it's own body or that of an opponent.

The objective of the game is to be the last snake alive, generating interesting

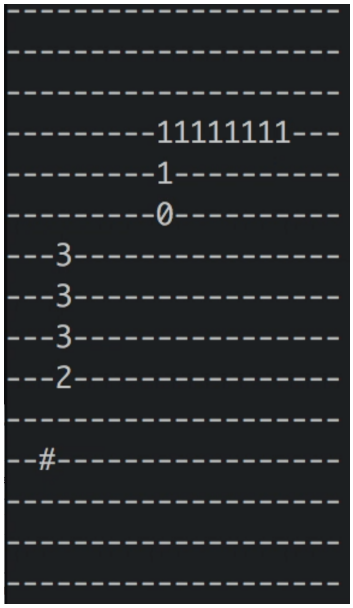

Figure 6-2: Ascii rendering of the multiplayer snake game.

behaviors where players compete to grow in size and try to trap the opponent in closed spaces. The game currently supports an unlimited number of snakes and arbitrary dimensions.

State

A structured state representation including all that is visible to a human player was included as the game is very simple. The state is represented by a $H * W$ integer matrix, which for a small scenario of a 20x20 grid consists of 400 features, which is easily treatable with a Convolutional Neural Network. Each cell can have several possible values:

- **1**: if the cell is empty.
- **2**: if the cell contains a fruit.
- **2*snake + 3**: if the cell contains the snake's head, where snake is the 0-indexed snake number.
- **2*snake + 4**: if the cell contains the snake's body.

A one-hot encoding state representation was considered as well, however it was dismissed as games with many players would have a considerably sparse representation, especially for larger grids. This hypothesis remains to be tested in a training scenario.

The state can be accessed with the g*et_state* and g*et_state_array* methods in the S*nakeGameEngine* class.

Actions

The game has a very reduced set of actions to simplify the training. They are stored in an enum called A*ction* and can be obtained by calling g*et_minimal_action_set* in the S*nakeGameEngine* class. Only three Actions are available:

- **NO_OP**: no change in direction.
- **LEFT**: left turn relative to the snake's head.
- **RIGT**: right turn relative to the snake's head.

Rewards

The game has a discrete set of rewards. They are returned after each call to the m*ove* method in the form of a vector, with one entry per snake. The values are the following:

- **1**: if the snake eats a fruit.
- **-1**: if the snake dies.
- **0**: otherwise.

Agent Implementation

Using the ELF framework, the Advantage Actor Critic algorithm was used to benchmark the environment training performance. The raw contents of the game state

described were used as the state representation, the playing field's dimension was of 10x10 cells.

A convolutional Neural Network consisting of two convolutional layers of 16 3x3 filters with leaky rectified linear unit activations was used. The result was flattened into a single feature vector of 1152 units, followed by a fully connected layer of 512 units. Finally the policy branch is created, fully connecting 3 units (number of moves) with the previous layer, a value branch is also created, connecting the previous layer with a single output unit.

Simulation Results

The performance of simulating the game in the ELF framework was measured by running a batch of 50000 games with 64 threads using a different number of cores. Each simulation was repeated 10 times and the average obtained. The simulation used a single player executing random moves and no training loop. The results obtained using an Intel Xeon 2.6GHz (x86_64) with 16 cores and 120 GB of RAM were the following:

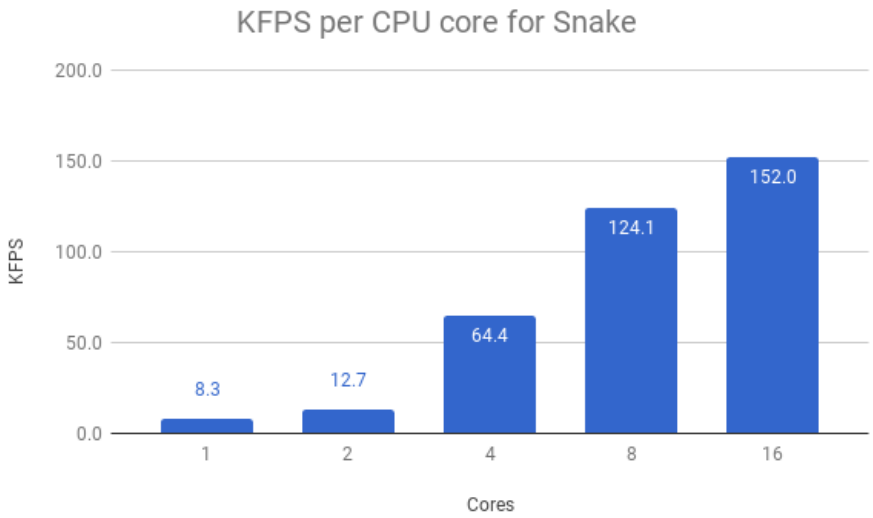

Figure 6-3: KFPS per core.

The results are very positive when compared to other classical "toy" games, such

as the Arcade Learning Environment [28]. The original ELF paper [25] cites a maximum performance of 5 KFPS when using a maximum of 16 cores and 64 threads, compared to 152 in multiplayer snake. Speed is an advantage when creating an introductory environment oriented for beginners who don't usually have access to large computational resources.

6.2 Future work

- **Contributing the multiplayer snake environment to the ELF repository**: the project needs a few small style tweaks and documentation before a merge request can be made. The game could be a great addition to the set of games in ELF, and a very good one for beginners who are looking for a simple problem to get started in self-play reinforcement learning. This could also encourage more people trying the challenge posed by OpenAI requests for research.

- **Write guide detailing how to implement new games into ELF**: the barrier of entry to integrating this game was very high. The main reason is there is no guide or documentation on how to extend the game, and instead the developer needs to spend a significant amount of time understanding the codebase and learning how to put the pieces together. There are several non-standard patterns such as directly sharing memory bytes from the C++ process into python. First a game in C++ needs to be implemented and binded to python via `PYBIND11`, if any problems arise in C++ such as segmentation faults, then a debugger like `gdb` needs to be used as the python interpreter crashes.

- **Use the environment to train a self-play reinforcement learning agent**: the original goal of this project was to train a self-play reinforcement learning agent. While the Pokémon problem turned out to be significantly more complex than expected, trying with a simpler game such as multiplayer snake is now

possible because of this work. The author would like to try with a simple Deep-Q learning version with two players in a small grid as a start.

Chapter 7

Conclusions

7.1 AlphaZero can't be used in Pokémon Battles due to turn atomicity, imperfect information and stochasticity

This project naively assumed AlphaZero could be used as a method to treat any two-player game, however Pokémon is unlike any two-player game. Each of it's distinctive properties make it very challenging to use AlphaZero - turn atomicity make it challenging to model in search tree form, but coupled with imperfect information the branching factor can explode to $>$ 300,000. Stochasticity only adds to the problem as state transitions can't be accurately predicted, thus increasing the branching factor even further, and therefore other methods have to be used.

7.2 Imperfect information and stochastic RL algorithms achieve good results on similar games

There are fields of reinforcement learning research that tackle similar problems, in particular imperfect information and stochastic reinforcement learning. Problems like Leduc Hold'em Poker (simplified Texas Hold'em) share the level of imperfect

information (albeit at a reduced state space), and researchers have developed deep reinforcement learning algorithms such as Fictitious Neural Play that achieve results close to Nash Equilibrium.

7.3 Few open RL platforms offer off the shelf support for self-play

In the search of an end-to-end platform to test simple self-play reinforcement learning algorithms, it was discovered that very few such libraries exist. The ELF research platform by Facebook was the only publicly available tool with built-in self-play support. However the tool was challenging to use as it is a complex high-performant system with little documentation available.

Bibliography

[1] Gerald Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2):181–199, 2002.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. pages 1–9, 2013.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[5] William Gaetz, Sudha K. Kessler, Tim P.L. Roberts, Jeffrey I. Berman, Todd J. Levy, Michelle Hsia, Deborah Humpl, Erin S. Schwartz, Sandra Amaral, Ben Chang, and Lawrence Scott Levin. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *Annals of Clinical and Translational Neurology*, pages 1–19, 2017.

[6] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. StarCraft II: A New Challenge for Reinforcement Learning. *arXiv*, 2017.

[7] OpenAI. OpenAI Gym, https://github.com/openai/gym, 2016.

[8] Nintendo 1996-2018 Pokemon Series, Game Freak. Pokemon Series, 1996-2018.

[9] Pokémon Showdown, https://pokemonshowdown.com/, 2018.

[10] Scott Lee and Julian Togelius. Showdown AI competition. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 191–198, 2017.

[11] Kush Khosla, Lucas Lin, and Calvin Qi. *Artificial Intelligence for Pokemon Showdown*. PhD thesis, Stanford University, 2017.

[12] Vavsum. pokemon_ai, https://github.com/vasumv/pokemon_ai, 2015.

[13] Raluca D. Gaina, Diego Pérez-Liébana, and Simon M. Lucas. General video game for 2 players: Framework and competition. *2016 8th Computer Science and Electronic Engineering Conference, CEEC 2016 - Conference Proceedings*, pages 186–191, 2017.

[14] Andrew Ng. DeepLearning.ai.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[17] Image Processig Group - UPC.

[18] StrategyWiki. Pokémon/Competitive battling – StrategyWiki, 2018.

[19] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent Complexity via Multi-Agent Competition. 2:1–12, 2017.

[20] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. 48, 2016.

[21] R. Duncan Luce and Howard Raiffa. *Games and Decisions: Introduction and Critical Survey (Dover Books on Mathematics)*. Dover Publications, 1989.

[22] Chen-Yu Wei, Yi-Te Hong, and Chi-Jen Lu. Online Reinforcement Learning in Stochastic Games. (Nips), 2017.

[23] Albert Xin Jiang. Mulitagent Reinforcement Learning in Stochastic Games with Continuous Action Spaces. pages 1–14, 2004.

[24] Johannes Heinrich and David Silver. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. 2016.

[25] Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C. Lawrence Zitnick. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[26] Yuandong Tian, Jerry Ma*, Qucheng Gong*, Shubho Sengupta, Zhuoyuan Chen, and C. Lawrence Zitnick. Elf opengo. https://github.com/pytorch/ELF, 2018.

[27] OpenAI. Requests For Ressearch 2.0, 2018.

[28] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.